



Chap. 2 The ARM Architecture

2.1 The Acorn RISC Machine

■ The Acorn RISC Machine

- ◆ developed at Acorn Computers Limited, of Cambridge, England, between 1983 and 1985
- ◆ 1990: Advanced RISC Machines Limited (ARM Limited)

■ In 1983

- ◆ 16-bit CISC microprocessors
 - were slower than standard memory parts
 - had instructions that took many clock cycles to complete (in some case, many hundreds of clock cycles), giving them very long interrupt latencies
 - commercial microprocessor projects had absorbed hundreds of man-years of design effort
- ◆ the Berkeley RISC I

ARM公司的起源

- 1978年12月5日，物理学家赫爾曼·豪澤（Hermann Hauser）和工程師(Chris Curry)，在英國劍橋創立了CPU(Cambridge Processing Unit)公司



Acorn Computer

- 1979年，CPU公司改名為Acorn計算機公司。



- ARM這個名字的由來 -Roger Wilson和Steve Furber設計了他們自己的第一代32位、6M Hz的處理器，用它做出了一台RISC指令集的計算機，簡稱ARM（Acorn RISC Machine）

ARMv1



ARM Holdings

- 1990年，Acorn 分割出 獨立子公司 – ARM (Advanced RISC Machine)
- ARM自己不製造晶片
- ARM 授權方式：ODM
(**O**riginal **D**esign **M**anufacturer)



ARM Ltd. headquarters

ARM Processor Core

■ Architecture

- ◆ Versions 1 and 2 – Acorn RISC, 26-bit address
- ◆ Version 3 – 32-bit address, CPSR, and SPSR
- ◆ Version 4 – half-word, Thumb
- ◆ Version 5 – BLX, CLZ and BRK instructions

■ Processor cores

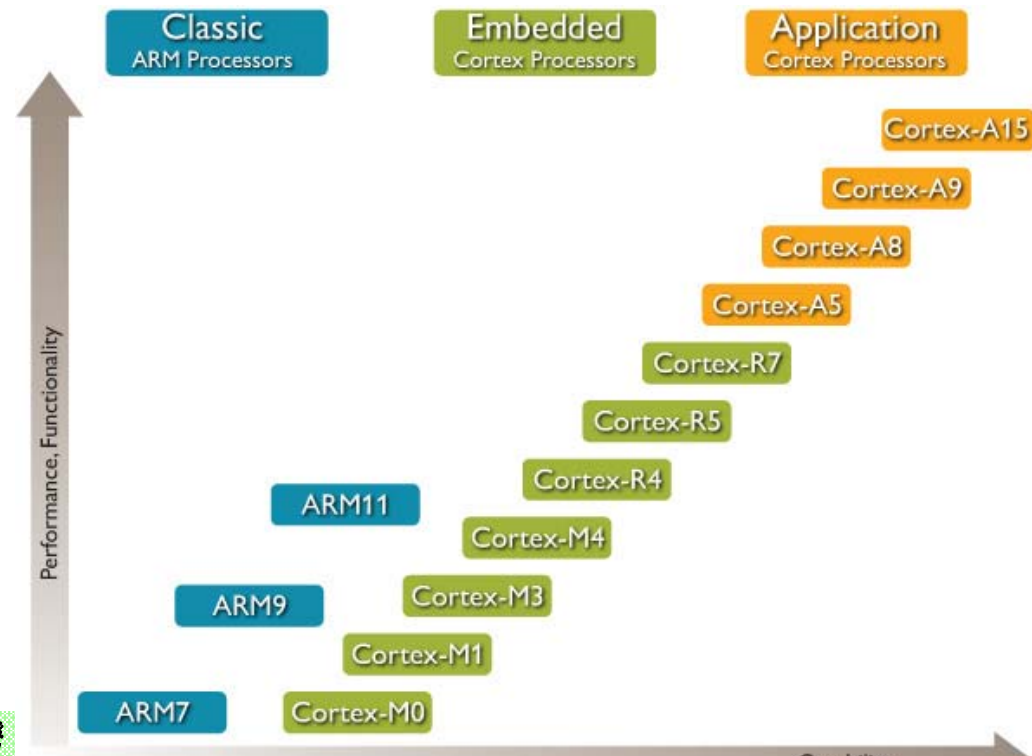
- ◆ ARM7TDMI (Thumb, debug, multiplier, ICE) – version 4T, low-end ARM core, 3-stage pipeline, 50-100MHz
- ◆ ARM9TDMI – 5-stage pipeline, 130MHz or 200MHz
- ◆ ARM10TDMI – version 5, 300MHz

■ CPU Core: co-processor, MMU, AMBA

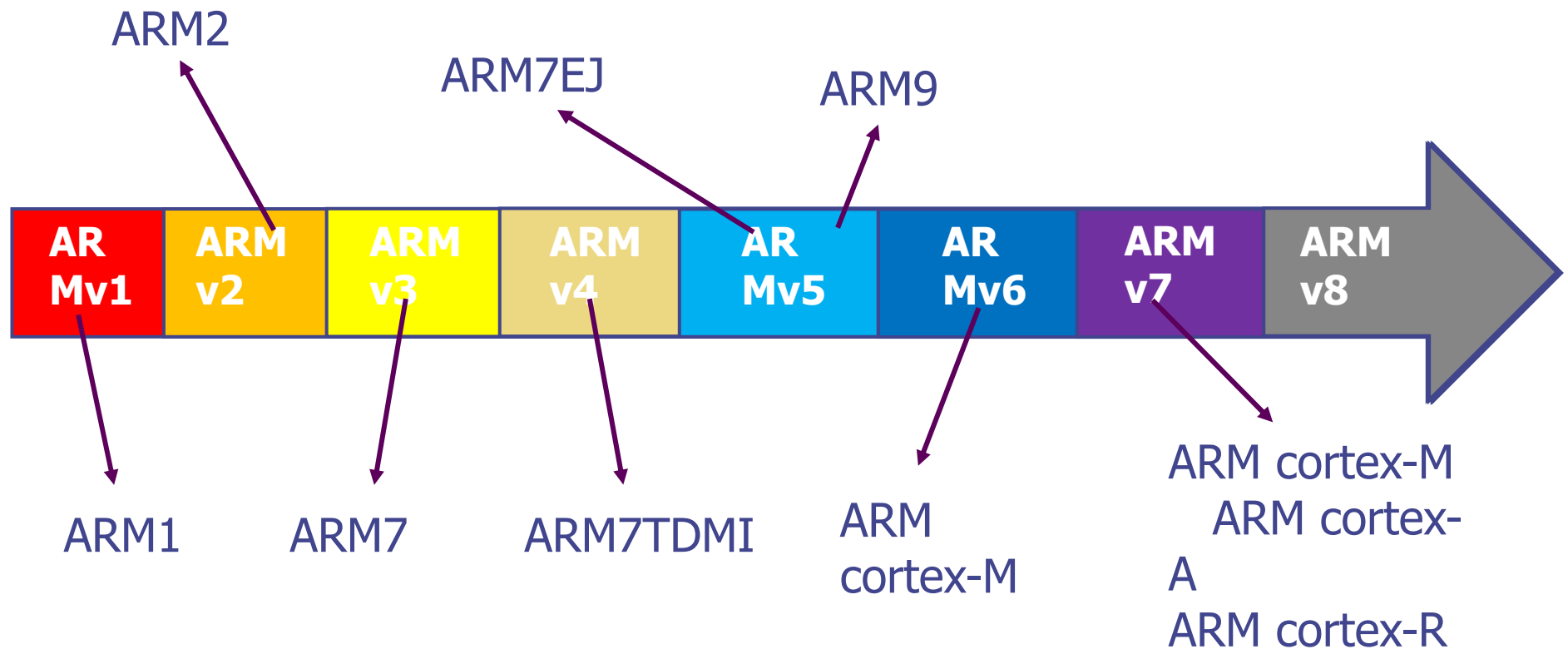
- ◆ ARM 710, 720, 740
- ◆ ARM 920, 940

Cortex-A、R、M

- Cortex-A的目標市場為32位元微處理器（A=Application Processor）
- Cortex-M為接續原有ARM7TDMI一路走來的微控器（M=MCU）
- Cortex-R是針對即時（R=Real Time）嵌入式應用需求的微控器，是一個關鍵任務（Mission Critical）用的應用



ARM內核演進



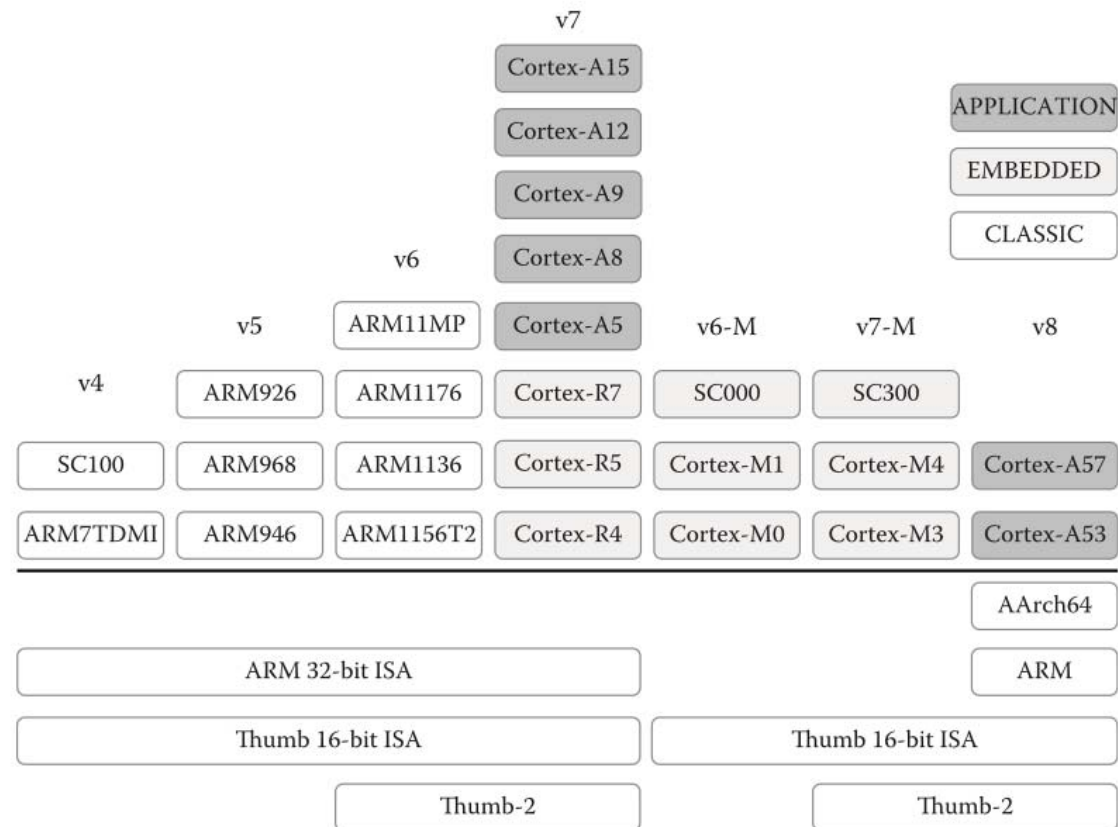


FIGURE 1.5 Architecture versions.

2.2 Architectural inheritance

- RISC architectures (at the first ARM chip was designed)
 - ◆ Berkeley RISC I and II, **Stanford MIPS**
- Features used (Berkeley RISC design)
 - ◆ **A load-store architecture**
 - ◆ **Fixed-length 32-bit instructions**
 - ◆ **3-address instruction formats**
- Features rejected
 - ◆ Register windows
 - The register banks on the Berkeley RISC processors incorporated a large number of registers, **32** of which were visible at any time
 - Procedure entry and exit instructions moved the visible ‘window’ to give each procedure access to new registers
 - The principal problem with register windows is the large chip area occupied by the large number of registers
 - This feature was therefore rejected on cost grounds

2.2 Architectural inheritance

■ Features rejected (Cont.)

◆ Delayed branches

- Branches cause pipelines problems since they interrupt the smooth flow of instructions
- Most RISC processors ameliorate the problem by using delayed branches where the branch takes effect after the following instruction has executed
- On the original ARM delayed branches were not used because they made exception handling more complex
- In the long run this has turned out to be a good decision since it simplifies re-implementing the architecture with a different pipeline

◆ Single-cycle execution of all instruction

- Although the ARM executes most data processing instructions in a single clock cycle, many other instructions take multiple clock cycles
- Single cycle operation of all instructions is only possible with separate data and instruction memories, which were considered too expensive for the intended ARM application areas
- The ARM was designed to use the minimum number of cycles required for memory access

Delayed branch's concept

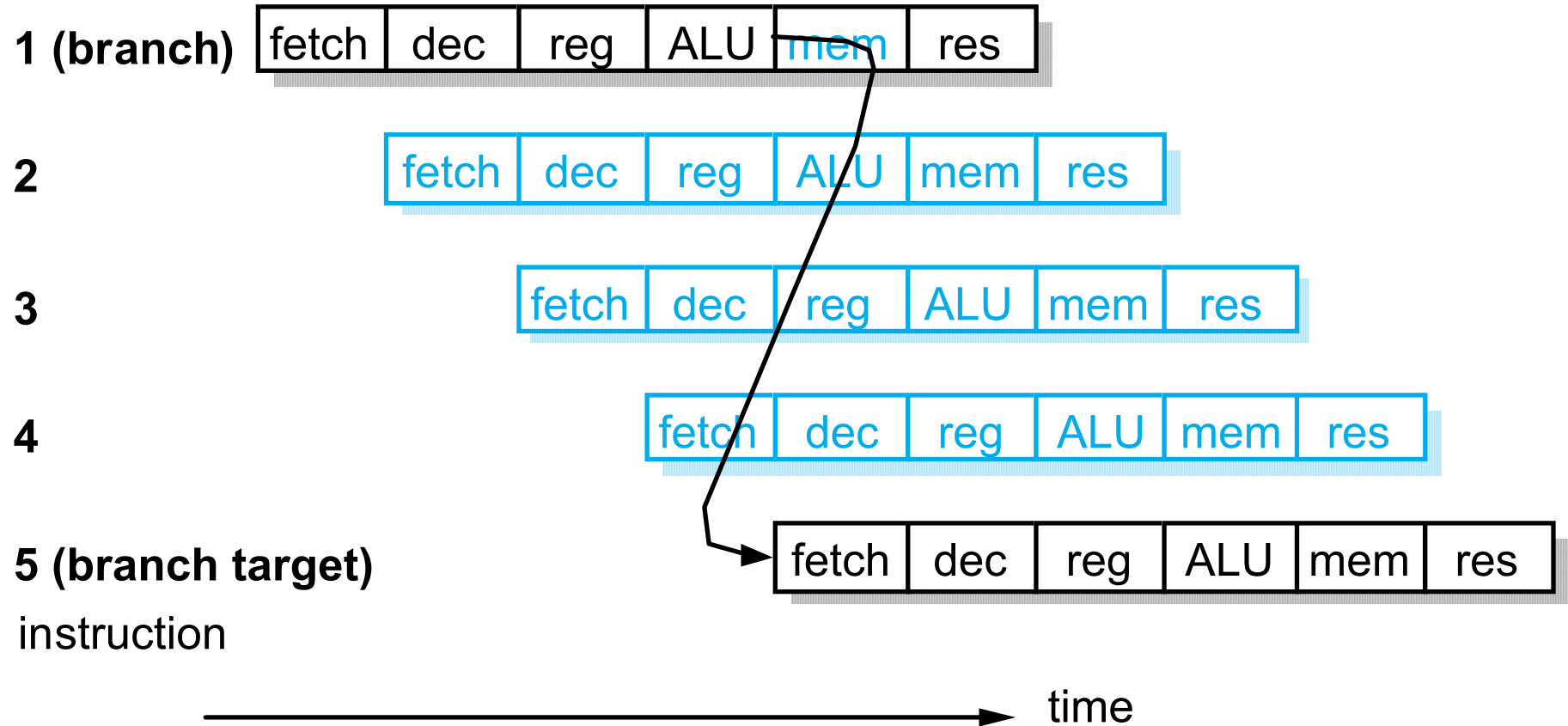


Fig. 1.15 Pipelined branch behaviour

2.2 Architectural inheritance

■ Simplicity

- ◆ An overriding concern of the original ARM design team was the need to keep the design **simple**
- ◆ The simplicity of the ARM may be more apparent in the **hardware organization and implementation** (Chap. 4) than it is in the instruction set architecture
- ◆ The combination of the simple hardware with an instruction set that is grounded in RISC ideas but **retains a few key CISC features**
 - achieve a significantly **better code density** than a pure RISC
 - give the ARM its power-efficiency and its small core size

2.3 The ARM programmer's model

■ The ARM programmer's model

- ◆ A processor's instruction set defines the operations the programmer can use to change the state of the system incorporating the processor
- ◆ This state usually comprises the values of the data items in the processor's visible registers and the system's memory
- ◆ Although a processor will typically have many invisible registers involved in executing an instruction, the values of these registers before and after the instruction is executed are not significant; only the values in the visible registers have any significance (Fig. 2.1)
- ◆ When writing user-level programs, only the 15 general-purpose 32-bit registers (r0 to r14), the program counter (r15) and the current program status register (CPSR) need be considered
- ◆ The remaining registers are used only for system-level programming and for handling exceptions (for example, interrupts)

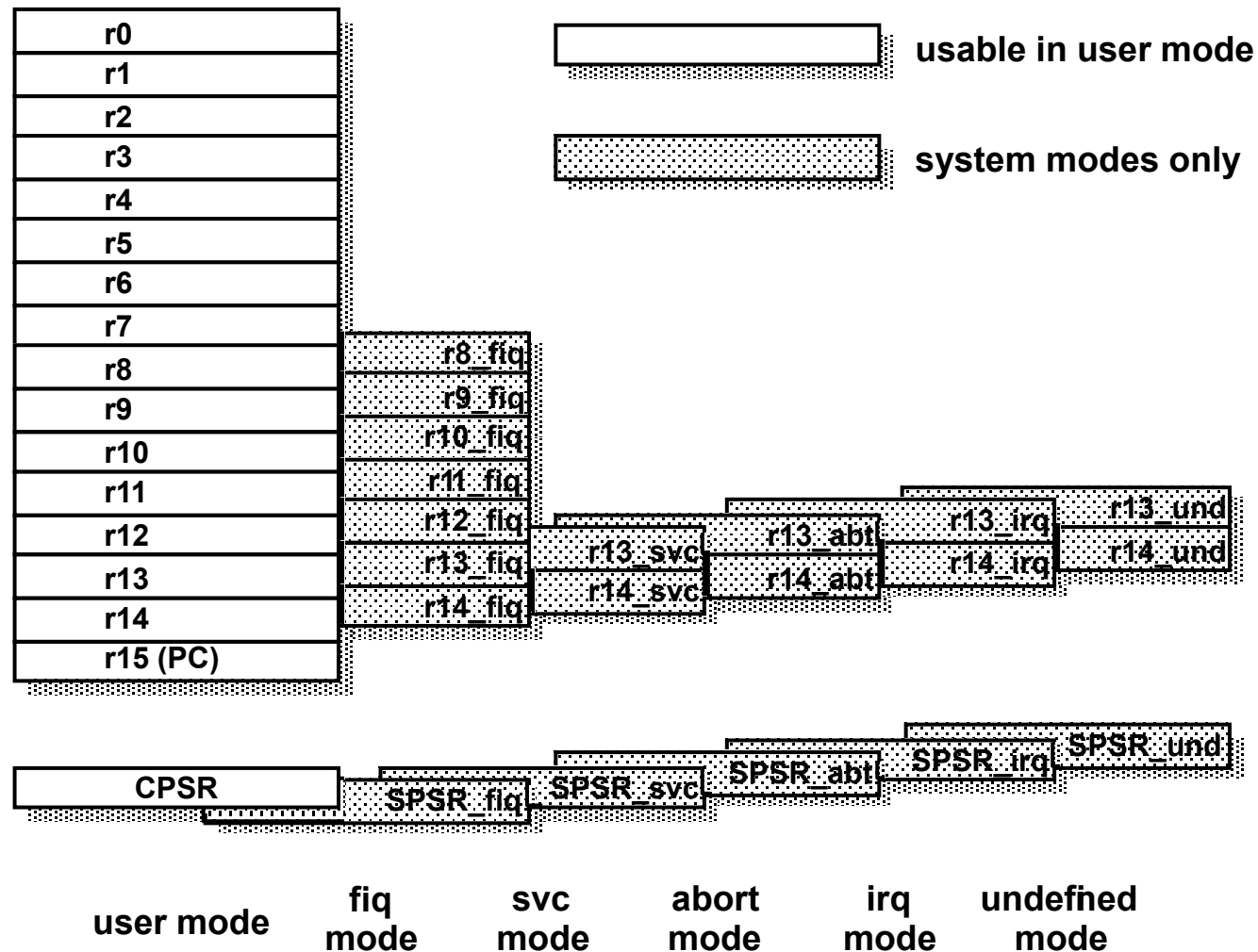


Fig. 2.1 ARM's visible registers

Register Organization

General registers and Program Counter

User32 / System	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13 (sp)	r13_fiq	r13_svc	r13_abt	r13_irq	r13_undef
r14 (lr)	r14_fiq	r14_svc	r14_abt	r14_irq	r14_undef
r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)

Program Status Registers

cpsr	cpsr spsr_fiq	cpsr spsr_svc	cpsr spsr_abt	cpsr spsr_irq	cpsr spsr_undef
------	------------------	------------------	------------------	------------------	--------------------



Exception modes

Mode	Description	Privileged modes
Supervisor (SVC)	Entered on reset and when a Software Interrupt (SWI) instruction is executed	
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a low priority (normal) interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	Unprivileged mode
User	Mode under which most applications/OS tasks run	

FIGURE 2.1 Processor modes.



Mode					
User/System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8_FIQ
R9	R9	R9	R9	R9	R9_FIQ
R10	R10	R10	R10	R10	R10_FIQ
R11	R11	R11	R11	R11	R11_FIQ
R12	R12	R12	R12	R12	R12_FIQ
R13	R13_SVC	R13_ABORT	R13_UNDEF	R13_IRQ	R13_FIQ
R14	R14_SVC	R14_ABORT	R14_UNDEF	R14_IRQ	R14_FIQ
PC	PC	PC	PC	PC	PC

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_SVC	SPSR_ABORT	SPSR_UNDEF	SPSR_IRQ	SPSR_FIQ


 = banked register

FIGURE 2.2 Register organization.

Accessing Registers using ARM Instructions

- No breakdown of currently accessible registers.
 - ◆ All instructions can access r0-r14 directly.
 - ◆ Most instructions also allow use of the PC.
- Specific instructions to allow access to CPSR and SPSR.
- Note : When in a privileged mode, it is also possible to load / store the (banked out) user mode registers to or from memory.
 - ◆ See later for details.

Processor Modes

■ The ARM has six operating modes:

- ◆ *User* (unprivileged mode under which most tasks run)
- ◆ *FIQ* (entered when a high priority (fast) interrupt is raised)
- ◆ *IRQ* (entered when a low priority (normal) interrupt is raised)
- ◆ *Supervisor* (entered on reset and when a Software Interrupt instruction is executed)
- ◆ *Abort* (used to handle memory access violations)
- ◆ *Undef* (used to handle undefined instructions)

■ ARM Architecture Version 4 adds a seventh mode:

- ◆ *System* (privileged mode using the same registers as user mode)

2.3 The ARM programmer's model

- The processor mode determines which registers are active and the access rights to the **cpsr** register.
 - ◆ A privileged mode allows full read-write access to the **cpsr**.
 - Abort: a failed attempt to access memory
 - fast interrupt request, interrupt request.
 - Supervisor: the mode the processor is in after reset.
 - System: Special version of user mode that allows full read-write access to **cpsr**.
 - Undefined: encounter an undefined mode.
 - ◆ A non-privileged mode only allows read access to the control field in the **cpsr** but still allows read-write access to the condition flags.
 - User: used for programs and application
- The processor mode can be changed by a program that writes directly to the **cpsr** or by hardware when the core responds to an exception or interrupt.
- Supervisor mode
 - ◆ The ARM processor supports a protected supervisor mode
 - ◆ The protection mechanism ensures that user code cannot gain supervisor privileges without appropriate checks being carried out to ensure that the code is not attempting illegal operations

2.3 The ARM programmer's model

■ The Current Program Status Register (CPSR)

- ◆ CPSR: is used in user-level programs to store the **condition code bits**
- ◆ The bits at the bottom of the register control the processor mode, instruction set, and interrupt enables and are protected from change by the user-level program (Fig. 2.2)
- ◆ The condition code flags are in the **top four bits** of the register and have the following meanings:
 - **N**: Negative (the top bit of the 32-bit result was a one)
 - **Z**: Zero (every bit of the 32-bit result was zero)
 - **C**: Carry (generate a carry-out)
 - **V**: oVerflow (generate an overflow into the sign bit)

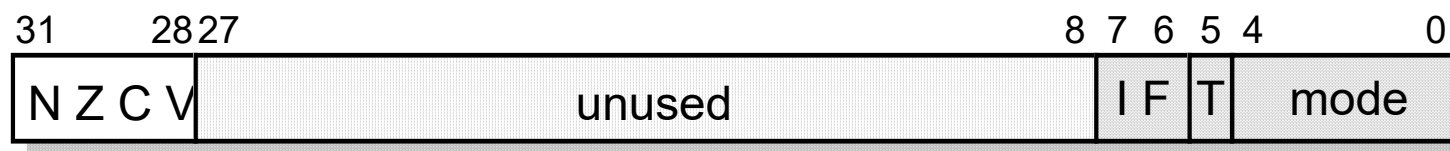


Fig. 2.2 ARM CPSR format

The Program Status Registers (CPSR and SPSRs)



Copies of the ALU status flags (latched if the instruction has the "S" bit set).

* **Mode Bits**
M[4:0] define the processor mode.

* Condition Code Flags

N = Negative result from ALU flag.
Z = Zero result from ALU flag.
C = ALU operation Carried out
V = ALU operation oVerflowed

* Interrupt Disable bits.

I = 1, disables the IRQ.
F = 1, disables the FIQ.

* T Bit (Architecture v4T only)

T = 0, Processor in ARM state
T = 1, Processor in Thumb state

M[4:0]	Mode
10000	User
10001	FIQ
10010	IRQ
10011	SVC
10111	Abort
11011	Undef
11111	System

Purpose of exception

- Provide an hardware-support system-level exception handling mechanism.
- Provide an interaction mechanism between processor (software) with other peripheral modules (hardware).
- Provide an system-function call mechanism.

Exception Handling and the Vector Table

■ When an exception occurs, the core:

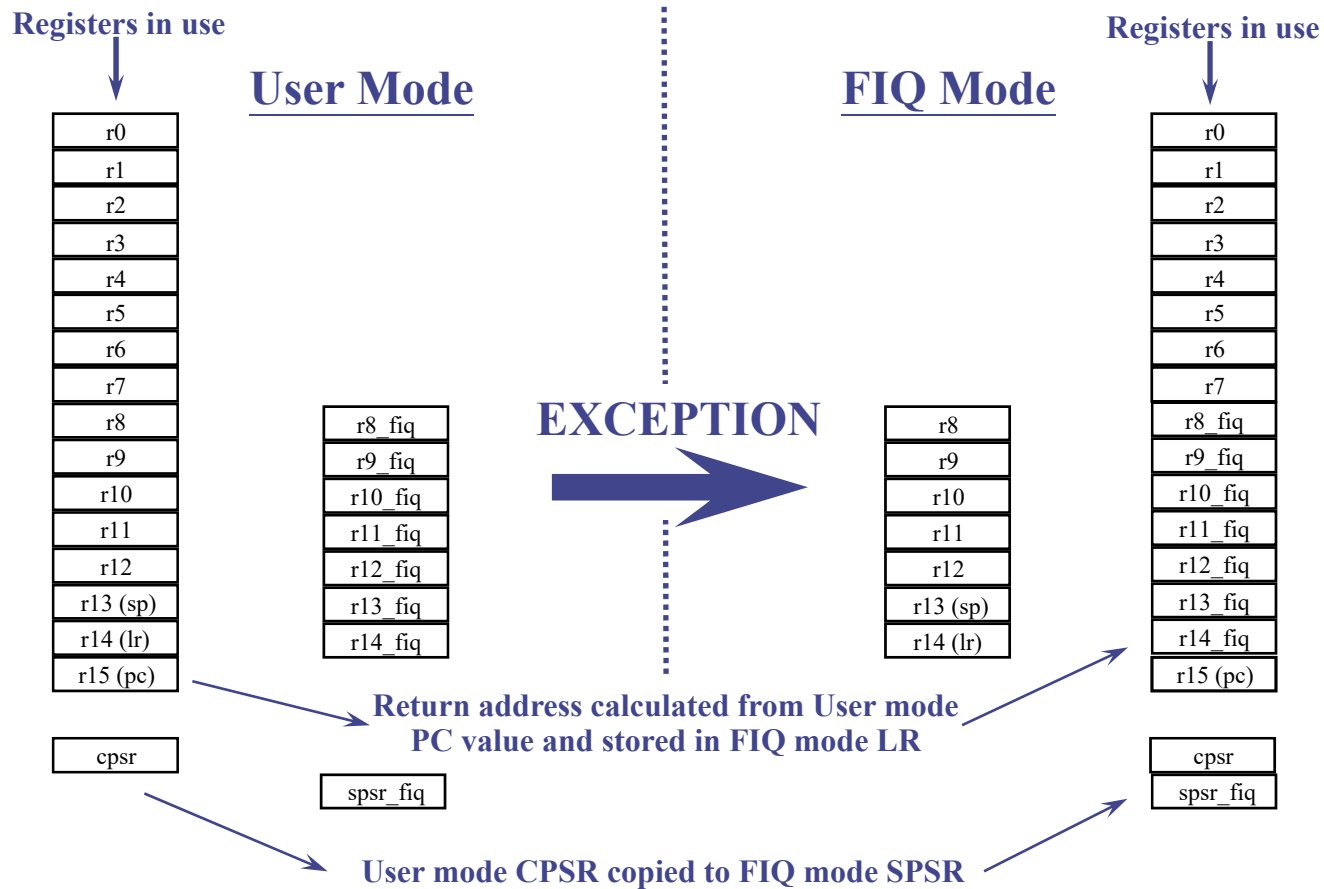
- ◆ Copies CPSR into SPSR_<mode>
- ◆ Sets appropriate CPSR bits
 - u If core implements ARM Architecture 4T and is currently in Thumb state, then
 - ARM state is entered.
 - u Mode field bits
 - u Interrupt disable flags if appropriate.
- ◆ Maps in appropriate banked registers
- ◆ Stores the “*return address*” in LR_<mode>
- ◆ Sets PC to vector address

0x00000000	Reset
0x00000004	Undefined Instruction
0x00000008	Software Interrupt
0x0000000C	Prefetch Abort
0x00000010	Data Abort
0x00000014	Reserved
0x00000018	IRQ
0x0000001C	FIQ

■ To return, exception handler needs to:

- ◆ Restore CPSR from SPSR_<mode>
- ◆ Restore PC from LR_<mode>

Register Example: User to FIQ Mode



2.3 The ARM programmer's model

■ ARM exceptions (Section 5.2)

- ◆ Interrupts, traps and supervisor calls
 - all grouped under the general heading of exceptions
- ◆ The general way
 - current state is saved by coping PC into r14_exc and CPSR into SPSR_exc (exc: exception type, Fig. 2.1)
 - processor operating mode is changed to the appropriate exception mode
 - PC is forced to a value between 00₁₆ and 1C₁₆, the particular value depending on the type of exception
- ◆ The instruction at the location (*the vector address*) will generally contain a branch (B<address>, LDR pc, [pc, #offset]) to the exception handler which use r13_exec (*stack pointer*) to save some user registers for use as work registers.

2.3 The ARM programmer's model

■ The memory system

- ◆ The minimum size of data that can be indexed is a byte (8-bit).
- ◆ Memory may be viewed as a linear array of **bytes** numbered from zero up to $2^{32}-1$
- ◆ Data items may be 8-bit bytes, 16-bit half-words or 32-bit words
- ◆ A **word-sized** data item must occupy a group of four byte locations starting at a byte address which is a **multiple of four**
- ◆ **Half-words** occupy two byte locations starting at an **even byte** address
- ◆ ARM adopts the standard '**little-endian**' memory organization, but can also be configured to work with a '**big-endian**' memory organization.

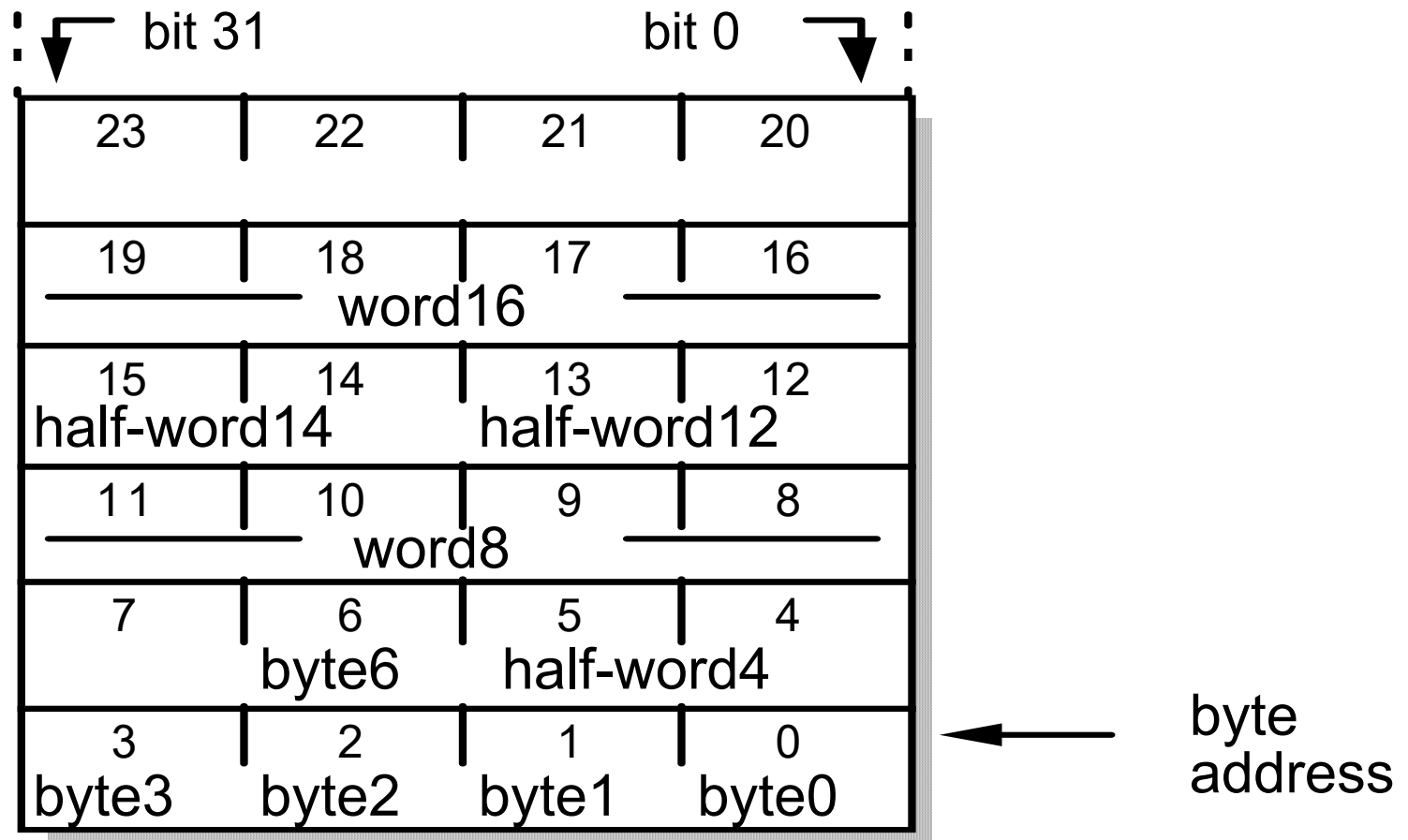
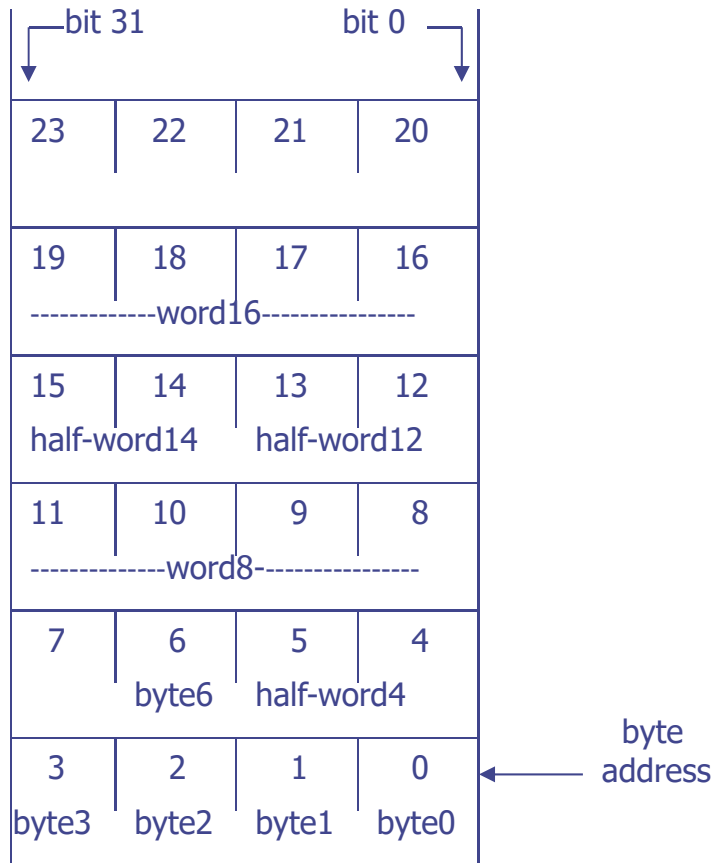
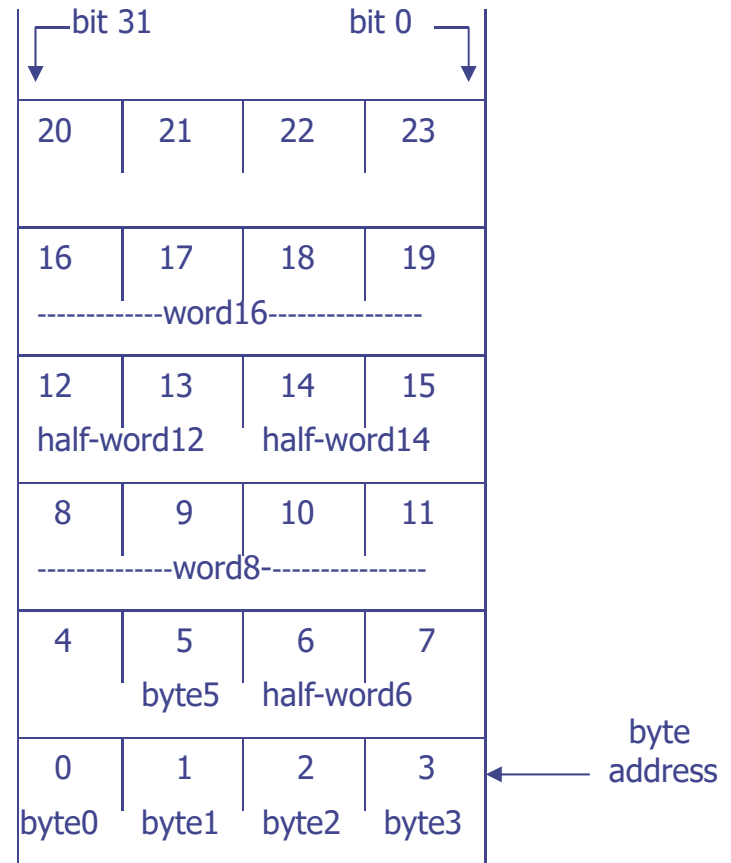


Fig. 2.3 ARM memory organization (Little-endian)

Memory Organization



(a) Little-endian memory organization



(b) Big-endian memory organization

2.3 The ARM programmer's model

■ Load-store architecture

- ◆ This means that the instruction set will only process (add, subtract, and so on) values which are in registers, and will always place the results of such processing into a register
- ◆ The only operations which apply to memory state are ones which copy memory values into registers (**load** instruction) or copy register values into memory (**store** instruction)
- ◆ All ARM instructions fall into one of the following three categories
 - **Data processing instruction**: use and change only register values
 - **Data transfer instructions**: copy memory values into registers or copy register values into memory
 - **Control flow instructions**: cause execution to switch to a different address either permanently (branch instruction), saving a return address to resume the original sequence (branch and link instruction) or trapping into system code (supervisor calls).

2.3 The ARM programmer's model

■ The ARM instruction set

- ◆ The load-store architecture
- ◆ 3-address data processing instructions
- ◆ Conditional execution of every instruction
- ◆ The inclusion of very powerful load and store multiple register instructions
- ◆ The ability to perform a general shift operation and a general ALU operation in a single instruction that executes in a single cycle
- ◆ A very dense 16-bit compressed representation of the instruction set in the Thumb architecture

2.3 The ARM programmer's model

■ The I/O system

- ◆ The ARM handles I/O peripherals (disk controllers, network interfaces, ...) as **memory-mapped devices** with interrupt support
- ◆ The **internal registers** in these devices appear as addressable locations within the ARM's memory map and may be read and written using the same instructions as any other memory locations
- ◆ Peripherals may attract the processor's attention by making an **interrupt** request
- ◆ Some systems may include **direct memory access** (DMA) hardware external to the processor to handle high-bandwidth I/O traffic (Section 11.9)

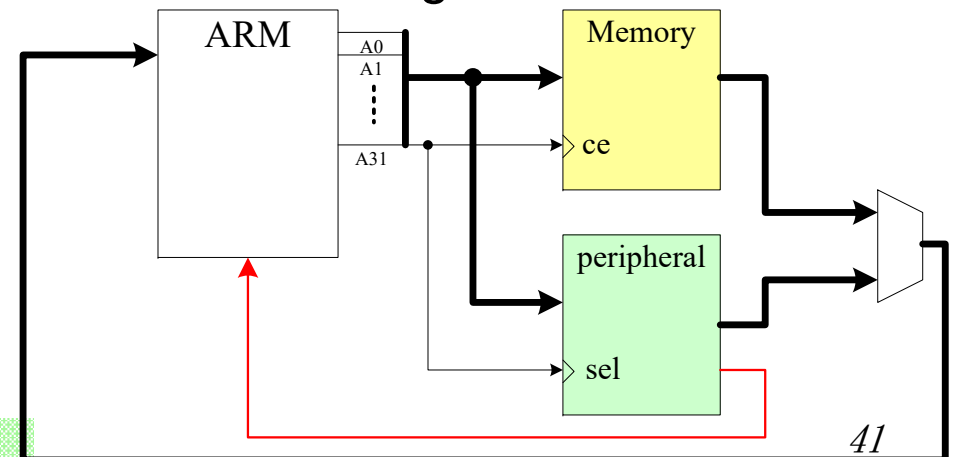




TABLE 5.1
Memory Map of the Tiva TM4C123GH6ZRB

Start	End	Description	For Details, See Page... ^a
Memory			
0x0000.0000	0x0003.FFFF	On-chip flash	553
0x0004.0000	0x00FF.FFFF	Reserved	—
0x0100.0000	0x1FFF.FFFF	Reserved for ROM	538
0x2000.0000	0x2000.7FFF	Bit-banded on-chip SRAM	537
0x2000.8000	0x21FF.FFFF	Reserved	—
0x2200.0000	0x220F.FFFF	Bit-band alias of bit-banded on-chip SRAM starting at 0x2000.0000	537
0x2210.0000	0x3FFF.FFFF	Reserved	—



Start	End	Description	For Details, See Page... ^a
Peripherals			
0x4000.0000	0x4000.0FFF	Watchdog timer 0	798
0x4000.1000	0x4000.1FFF	Watchdog timer 1	798
0x4000.2000	0x4000.3FFF	Reserved	—
0x4000.4000	0x4000.4FFF	GPIO Port A	675
0x4000.5000	0x4000.5FFF	GPIO Port B	675
0x4000.6000	0x4000.6FFF	GPIO Port C	675
0x4000.7000	0x4000.7FFF	GPIO Port D	675
0x4000.8000	0x4000.8FFF	SSI 0	994
0x4000.9000	0x4000.9FFF	SSI 1	994
0x4000.A000	0x4000.AFFF	SSI 2	994
0x4000.B000	0x4000.BFFF	SSI 3	994
0x4000.C000	0x4000.CFFF	UART 0	931
0x4000.D000	0x4000.DFFF	UART 1	931
0x4000.E000	0x4000.EFFF	UART 2	931
0x4000.F000	0x4000.FFFF	UART 3	931
0x4001.0000	0x4001.0FFF	UART 4	931
0x4001.1000	0x4001.1FFF	UART 5	931
0x4001.2000	0x4001.2FFF	UART 6	931
0x4001.3000	0x4001.3FFF	UART 7	931
0x4001.4000	0x4001.FFFF	Reserved	—



Start	End	Description	For Details, See Page... ^a
Peripherals			
0x4002.0000	0x4002.0FFF	I ² C 0	1044
0x4002.1000	0x4002.1FFF	I ² C 1	1044
0x4002.2000	0x4002.2FFF	I ² C 2	1044
0x4002.3000	0x4002.3FFF	I ² C 3	1044
0x4002.4000	0x4002.4FFF	GPIO Port E	675
0x4002.5000	0x4002.5FFF	GPIO Port F	675
0x4002.6000	0x4002.6FFF	GPIO Port G	675
0x4002.7000	0x4002.7FFF	GPIO Port H	675
0x4002.8000	0x4002.8FFF	PWM 0	1270



Start	End	Description	For Details, See Page... ^a
0x4002.9000	0x4002.9FFF	PWM 1	1270
0x4002.A000	0x4002.BFFF	Reserved	—
0x4002.C000	0x4002.CFFF	QE1 0	1341
0x4002.D000	0x4002.DFFF	QE1 1	1341
0x4002.E000	0x4002.FFFF	Reserved	—
0x4003.0000	0x4003.0FFF	16/32-bit Timer 0	747
0x4003.1000	0x4003.1FFF	16/32-bit Timer 1	747
0x4003.2000	0x4003.2FFF	16/32-bit Timer 2	747
0x4003.3000	0x4003.3FFF	16/32-bit Timer 3	747
0x4003.4000	0x4003.4FFF	16/32-bit Timer 4	747
0x4003.5000	0x4003.5FFF	16/32-bit Timer 5	747
0x4003.6000	0x4003.6FFF	32/64-bit Timer 0	747
0x4003.7000	0x4003.7FFF	32/64-bit Timer 1	747



Start	End	Description	For Details, See Page... ^a
0x4003.8000	0x4003.8FFF	ADC 0	841
0x4003.9000	0x4003.9FFF	ADC 1	841
0x4003.A000	0x4003.BFFF	Reserved	—
0x4003.C000	0x4003.CFFF	Analog Comparators	1240
0x4003.D000	0x4003.DFFF	GPIO Port J	675
0x4003.E000	0x4003.FFFF	Reserved	—
0x4004.0000	0x4004.0FFF	CAN 0 Controller	1094
0x4004.1000	0x4004.1FFF	CAN 1 Controller	1094
0x4004.2000	0x4004.BFFF	Reserved	—
0x4004.C000	0x4004.CFFF	32/64-bit Timer 2	747
0x4004.D000	0x4004.DFFF	32/64-bit Timer 3	747
0x4004.E000	0x4004.EFFF	32/64-bit Timer 4	747
0x4004.F000	0x4004.FFFF	32/64-bit Timer 5	747
0x4005.0000	0x4005.0FFF	USB	1146
0x4005.1000	0x4005.7FFF	Reserved	—



Start	End	Description	For Details, See Page... ^a
0x4005.8000	0x4005.8FFF	GPIO Port A (AHB aperture)	675
0x4005.9000	0x4005.9FFF	GPIO Port B (AHB aperture)	675
0x4005.A000	0x4005.AFFF	GPIO Port C (AHB aperture)	675
0x4005.B000	0x4005.BFFF	GPIO Port D (AHB aperture)	675
0x4005.C000	0x4005.CFFF	GPIO Port E (AHB aperture)	675
0x4005.D000	0x4005.DFFF	GPIO Port F (AHB aperture)	675
0x4005.E000	0x4005.EFFF	GPIO Port G (AHB aperture)	675
0x4005.F000	0x4005.FFFF	GPIO Port H (AHB aperture)	675
0x4006.0000	0x4006.0FFF	GPIO Port J (AHB aperture)	675
0x4006.1000	0x4006.1FFF	GPIO Port K (AHB aperture)	675
0x4006.2000	0x4006.2FFF	GPIO Port L (AHB aperture)	675
0x4006.3000	0x4006.3FFF	GPIO Port M (AHB aperture)	675



Start	End	Description	For Details, See Page... ^a
0x4006.4000	0x4006.4FFF	GPIO Port N (AHB aperture)	675
0x4006.5000	0x4006.5FFF	GPIO Port P (AHB aperture)	675
0x4006.6000	0x4006.6FFF	GPIO Port Q (AHB aperture)	675
0x4006.7000	0x400A.EFFF	Reserved	—
0x400A.F000	0x400A.FFFF	EEPROM and Key Locker	571
0x400B.0000	0x400B.FFFF	Reserved	—
0x400C.0000	0x400C.0FFF	I ² C 4	1044
0x400C.1000	0x400C.1FFF	I ² C 5	1044
0x400C.2000	0x400E.8FFF	Reserved	—
0x400F.9000	0x400F.9FFF	System Exception Module	497
0x400F.A000	0x400F.BFFF	Reserved	—
0x400F.C000	0x400F.CFFF	Hibernation Module	518
0x400F.D000	0x400F.DFFF	Flash memory control	553
0x400F.E000	0x400F.EFFF	System control	237
0x400F.F000	0x400F.FFFF	μDMA	618
0x4010.0000	0x41FF.FFFF	Reserved	—
0x4200.0000	0x43FF.FFFF	Bit-banded alias of 0x4000.0000 through 0x400F.FFFF	—
0x4400.0000	0xDFFF.FFFF	Reserved	—



Start	End	Description	For Details, See Page... ^a
Private Peripheral Bus			
0xE000.0000	0xE000.0FFF	Instrumentation Trace Macrocell (ITM)	70
0xE000.1000	0xE000.1FFF	Data Watchpoint and Trace (DWT)	70
0xE000.2000	0xE000.2FFF	Flash Patch and Breakpoint (FPS)	70
0xE000.3000	0xE000.DFFF	Reserved	—
0xE000.E000	0xE000.EFFF	Cortex-M4F Peripherals (SysTick, NVIC, MPU, FPU and SCB)	134
0xE000.F000	0xE003.FFFF	Reserved	—
0xE004.0000	0xE004.0FFF	Trace Port Interface Unit (TPIU)	71
0xE004.1000	0xE004.1FFF	Embedded Trace Macrocell (ETM)	70
0xE004.2000	0xFFFF.FFFF	Reserved	—

^a See Tiva TM4C123GH6ZRB Microcontroller Data Sheet.

2.4 ARM development tools

■ ARM development tools

- ◆ Since the ARM is widely used as an embedded controller where the target hardware will not make a good environment for software development, the tools are intended for cross-development from a platform such as a PC running Windows or a suitable UNIX workstation
- ◆ The overall structure of the ARM cross-development toolkit is shown in Fig. 2.4
- ◆ C or assembler source files are compiled or assembled into ARM **object format** (.aof) files, which are then linked into ARM image format (.aif) files.
- ◆ The **image format** files can be built to include the debug tables required by the ARM symbolic debugger (ARMsd which can load, run and debug programs either on hardware such as the ARM Development Board or using a software emulation of the ARM)

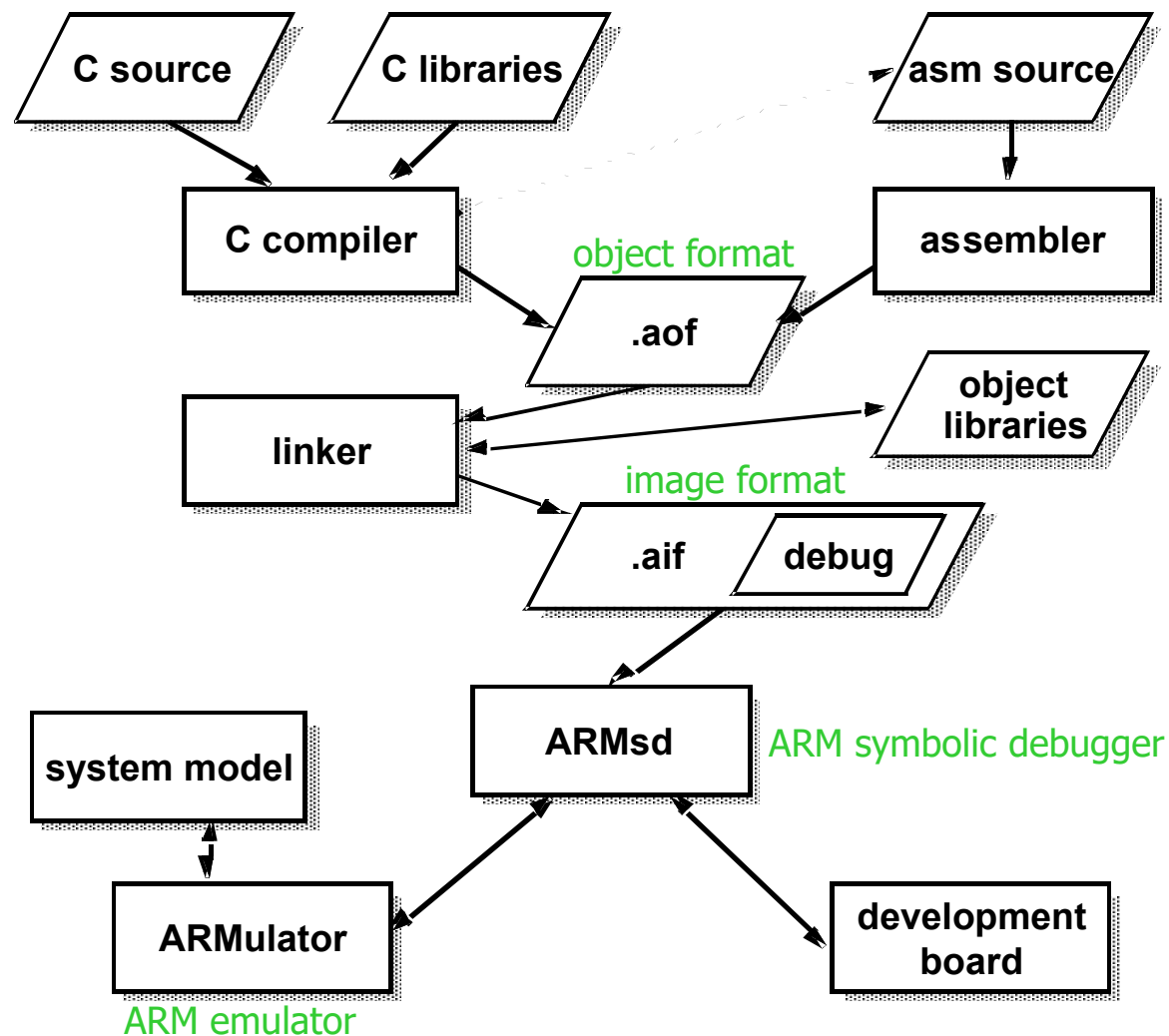


Fig. 2.4 The structure of the ARM cross-development toolkit

2.4 ARM development tools

■ The ARM C compiler

- ◆ The ARM C compiler is compliant with the ANSI standard for C and is supported by the appropriate library of standard functions
- ◆ It can be told to produce assembly source output instead of ARM object format, so the code can be inspected, or even hand optimized, and then assembled subsequently
- ◆ The compiler can also produce Thumb code

■ The ARM assembler

- ◆ The ARM assembler is a full macro assembler which produces ARM object format output that can be linked with output from the C compiler

2.4 ARM development tools

■ The linker

- ◆ The linker takes one or more object files and combines them into an executable program
- ◆ It resolves symbolic references between the object files and extracts object modules from libraries as needed by the program
- ◆ The linker can also produce object library modules that are not executable but are ready for efficient linking with object files in the future

2.4 ARM development tools

■ ARMsd

- ◆ The **ARM symbolic debugger** is a front-end interface to assist in debugging programs running either under emulation or remotely on a target system such as the ARM development board
- ◆ ARMsd allows an executable program to be loaded into the **ARMulator** or a **development board** and run
- ◆ It allows the setting of **breakpoints**, which are addresses in the code that, if executed, cause execution to halt so that the processor state can be examined

2.4 ARM development tools

■ ARMulator

- ◆ The **ARMulator** (ARM emulator) is a suite of programs that models the behaviour of various ARM processor cores in software on a host system
- ◆ At its simplest, the ARMulator allows an ARM program developed using the C compiler or assembler to be tested and debugged on a host machine with no ARM processor connected
- ◆ At its most complex, the ARMulator can be used as the centre of a complete, timing-accurate, C model of the target system, with full details of the cache and memory management functions added

2.4 ARM development tools

■ ARM development board

- ◆ The **ARM Development Board** is a circuit board incorporating a range of components and interfaces to support the development of ARM-based systems
- ◆ It includes an ARM core, memory components which can be configured to match the performance and bus-width of the memory in the target system, and electrically programmable devices which can be configured to emulate application-specific peripherals